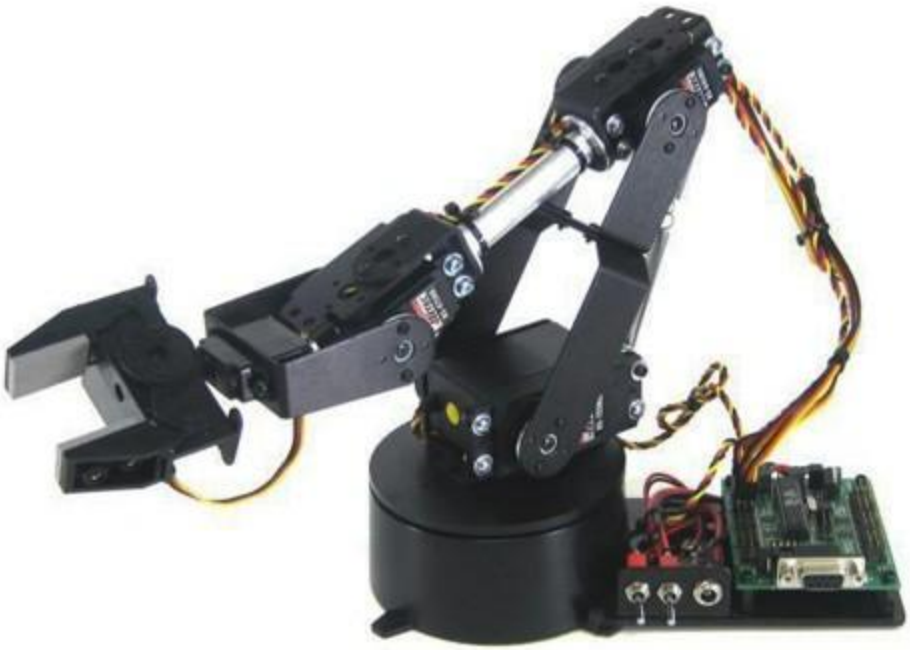
Projet 4ème année: Bras Robotique
Contrôle du mouvement



TIAMIOU Adam
BEN YOUNES Radhoane

POLYTECH GRENOBLE

Sommaire

I. Présentation du contexte……3

II- Travail effectué……..……4

1.Interface de connexion avec le robot ……...4
2.Récupération des données de la caméra…….5
3.Le simulateur……7

III- Ce qu'il reste à faire - Améliorations…….13

IV- Annexes……...14

I- Présentation du contexte.

Notre bras Robotique : le AL5B est équipé d'une carte SSC32. Cette carte permet une communication bidirectionnel et dispose de 32 canaux pouvant contrôler des servomoteurs. Celui-ci utilise une connexion série de 8 bits.

Le robot est aussi équipée d'une webcam qui lui permettra de détecter des objets grâce à des marqueurs.

Ce projet est divisé en deux partie :

- La première consiste à s'occuper de la détection du marqueur placé sur un objet.
- La seconde,qui est la notre, est de s'occuper de tout ce qui concerne la communication avec le robot et l'intelligence de ses mouvements pour atteindre des objectifs précis.

Pour notre projet, Nous avons fixé les objectifs suivant :

1.Le logiciel doit être capable de communiquer avec tous robots équipé d'une carte SSC-32

2. Notre logiciel doit être capable de calculer une trajectoire correct afin d'attraper l'objet.

3.-Pour connaître à tout moment les coordonnées de la pince. Un simulateur en Vpython doit être implémenté, représentant le bras robotique virtuellement.

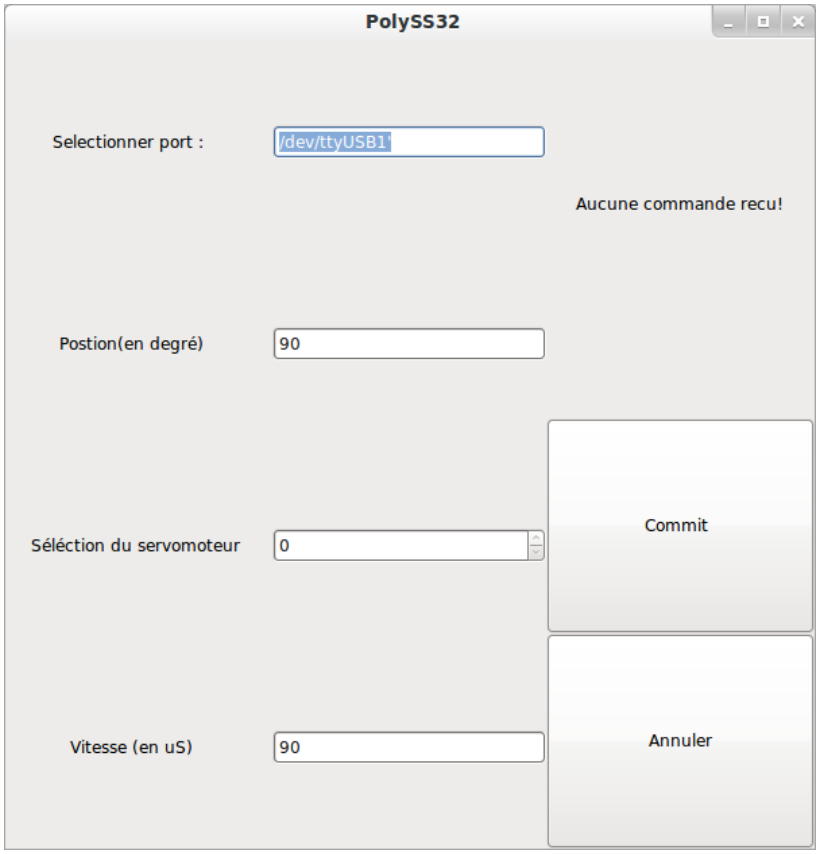
4.On doit être capable de communiquer avec le deuxième groupe afin de récupérer les informations sur la position du marqueur et les déplacement à effectuer.

II- Travail effectué.

Le projet est séparé en deux partie: La première concerne le développement d'une interface qui permet la connexion avec le robot, et la deuxième le simulateur ainsi que la communication avec la caméra.

1.Interface de connexion avec le robot .

Pour la première partie, il a fallut réfléchir sur le fonctionnement de la carte SSC-32 grâce à son manuel d'utilisation. Le but étant de définir et changer pour chaque servomoteur la valeur des angles afin d'être en accord avec les mouvement du simulateur robot.
Nous avons donc défini deux classes(voir annexe) ainsi qu'une interface graphique qui va nous permettre d'effectuer des tests directement sur le robot:



Lorsqu'un on clique sur le bouton commit, on voit afficher la commande qui va être envoyé au robot qui est définit selon les différentes options présentes: Vitesse,Position...

Par exemple si on clique sur commit on vois affiché : `#0P918S90 <cr>`.
Remarque : la position en degrés est automatiquement traduite en sa valeur en pulsation.

Par défaut, la connexion est désactivée pour pouvoir afficher les valeurs sans avoir être connecter aurobot. Il suffit de dé-commenter la fonction suivante ***ss.write(valeur)*** dans la méthode on_my_mouse_clicked2.

Nous avons aussi crée un autre fichier (test.py dans le document connexion, voir annexe) qui permet de tester le déplacement de plusieurs servomoteurs simultanément et cela grâce à une liste de position prédéfinie pour chaque servomoteur.

2.Récupération des données de la caméra:

Pour pouvoir communiquer avec le groupe s'occupant de la partie détection d'image, nous avons opté pour un système de communication Client/Serveur en utilisant des Agents (contenu dans le dossier Agent).
Une fois le serveur lancé, si la caméra détecte le marqueur, un fichier ".xml" va être crée.
Il contient les informations suivante qui sont les coordonnées des quatre coins du marqueur :

<p1 x="236.551" y="230.127"/>
<p2 x="239.44" y="176.346"/>
<p3 x="293.506" y="178.652"/>
<p4 x="290.598" y="232.49"/>

Ensuite nous récupérons les coordonnées de l'objet **par rapport à l'image** grâce à la fonction " xmldoc = minidom.parse('fichier.xml')".

Une fois les données reçues, on peut les afficher sur un plan xOy. Le but étant qu'à partir de ces coordonnées et en connaissant la taille de la fenêtre, nous puissions centrer le marqueur par rapport au centre de la fenêtre, comme précisé dans la partie simulateur (Page 7).

On a donc défini un intervalle dans lequel les coordonnées doivent être comprise pour que le marqueur puisse être considéré comme centré.

Ensuite, on effectue des mouvements guidés par la caméra. Par exemple si le marqueur est sur le coin supérieur droite, le robot doit bouger plus à gauche et vers le bas. Jusqu'à ce qu'il soit compris dans l'intervalle définit précédemment.

Une fois l'objet centrer, il reste un petit détail :

On sait que du moment où le marqueur apparaît sur l'écran, il devient facile de s'approcher de lui, car on a juste à considérer les déplacements qui gardent celui-ci en vue. Cependant, on ne voyait pas comment un simulateur en 3D qui a besoin des coordonnées x,y,z, pouvait attraper un objet sans cette dernière composante.
En effet, il est impossible d'obtenir la position d'un objet en coordonnées x,y,z en se servant de l'image fixe d'une seule caméra.

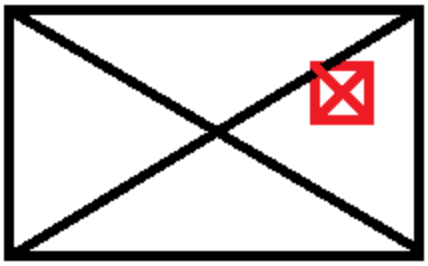
Après une discussion avec l'autre groupe, nous avons dégagé une manière intelligente de répondre à ce problème en considérant la taille initiale du marqueur. Une fois l'objet centré et droit, on effectue un déplacement de la caméra vers le marqueur et en fonction de la précédente coordonnées et de la nouvelle (du marqueur), on obtenait une bonne estimation de la profondeur (qui sépare la pince de l'objet), avec une marge d'erreur qui diminue à mesure qu'on s'approche.

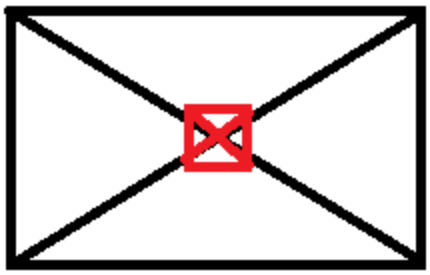
Nous avons mis cette méthode de côté en attendant que toutes les étapes précédentes soient accomplies, aussi bien dans l'autre groupe que dans le notre.

3.Simulateur

A) INTRODUCTION

Après avoir discuté avec le groupe s'occupant de la détection de marqueur, on a estimé que la caméra, positionnée sur la pince du robot (ou en amont), devait permettre au robot de recevoir des ordres en fonction de la position, du marqueur détecté, sur l'écran. Le but étant que le marqueur soit centré sur l'écran.





Cela est devenu concret lorsque le groupe a réussi à obtenir les coordonnées des quatre points du marqueur suivant l'origine qui se trouve au coin en haut à gauche.
On a donc une translation de la caméra qui bouge dans l'espace pour se centrer. Étant donné qu'elle est positionnée sur la pince, il est donc logique que c'est celle-ci qui doit effectuer ces translations. On doit donc exprimer la pince du robot en coordonnées indépendantes de la caméra.

La première question fût de déterminer un repère pour rendre explicite ces futures coordonnée. Logiquement, c'est le socle du robot ou le point de départ du premier bras qui servira de repère puisqu'il ne change jamais.

On décide donc de se servir de ce point pour déterminer toutes les coordonnées des points d'intersection des axes robotiques, en fonction de la longueur de chacun d'eux et des angles de départ.

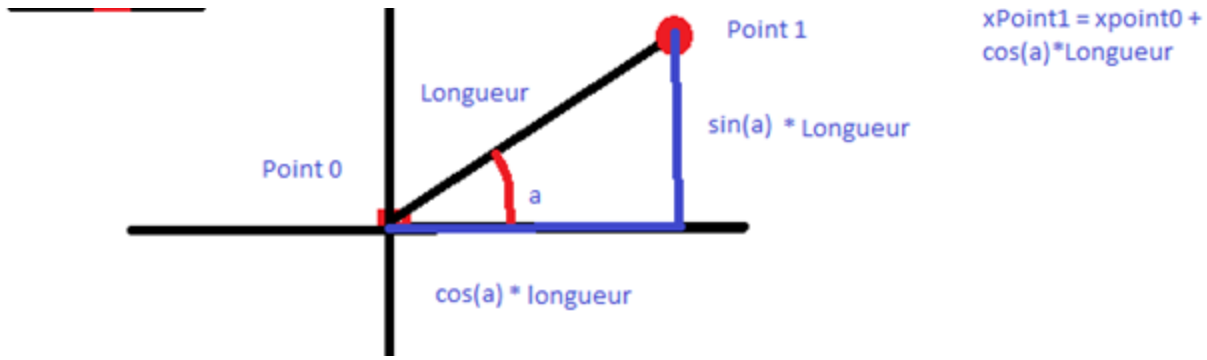
(De gauche à droite : Point 0, Point 1, Point 2, Point 3)

B) CALCUL SUR LE PLAN xOy

1°) COORDONNEES DES POINTS

Pour le moment, on se limite au plan xOy et on définit la même z que celui du socle à tous les points.
Cette même valeur de z pour chaque point va trouver tout son sens plus tard.
Mais pour l'instant, cette coordonnée n'apparait pas dans les calculs.

Pour calculer les coordonnées du point 1 :



Voici comment fonctionne VPython pour créer une boite entre deux points :
On doit donner la position du centre de la boite. On se sert donc du calcul du milieu d'un segment (x = (x0+x1) /2).
Pour trouver les points 2 et 3, on cherche le point qui se situe sur le vecteur vPoint1versPoint0 (Pour le cas du point 1) en conservant la taille du bras (bras 2 pour ce cas) et on effectue une rotation par rapport à l'angle donné en partant de ce point.

2°) ROTATION DES BRAS

Par la suite, on s'est intéressé à la modification de l'angle des bras suivant un angle donné.
Par exemple, si on modifie celui du bras 1, les points des autres bras vont aussi subir cette rotation.

Ainsi, grâce à une simple formule mathématique, on calcule les coordonnées du nouveau point :
XPoint1 = (xPoint1-xPoint0) * cos(angle) - (yPoint1-yPoint0) * sin(angle) +xPoint0
YPoint1 = (xPoint1-xPoint0) * sin(angle) +(yPoint1-yPoint0) * sin(angle) +xPoint0

On va se servir souvent de cette formule.

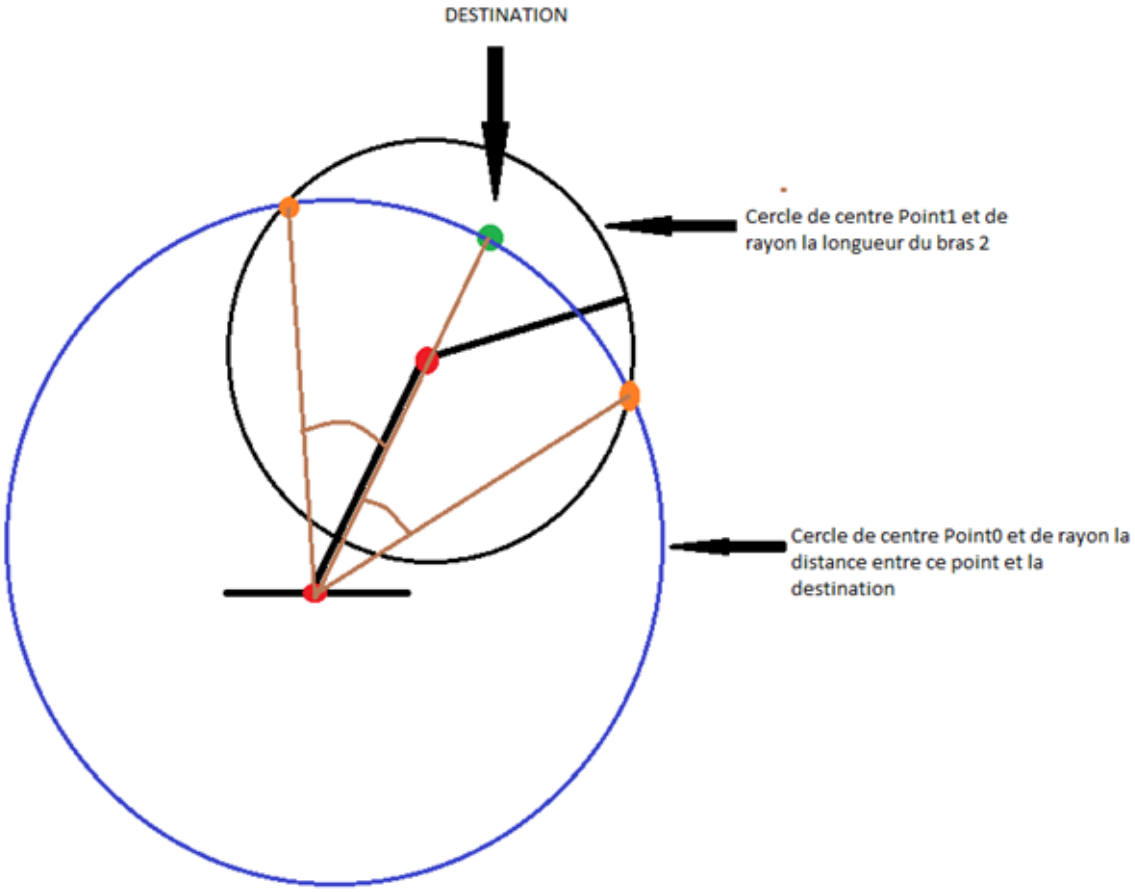
En gros, on fait une rotation de tous les points (sauf le socle bien sûr) si on bouge le bras1, des même points sauf le point1 si on bouge le bras 2 etc…
Afin d'anticiper l'utilisation du mode recherche de la pince, on décide de priver le point3 de rotation lorsque l'on bouge les bras 1 et 2. On effectue des translations qui suivent les déplacements du point 2.

3°) DEPLACEMENT DE LA PINCE VERS UNE DESTINATION

Une fois que le robot est capable de bouger suivant des angles, on est donc en mesure de passer au mouvement de la pince.
En gros, nous avons une destination et l'objectif est que la pince l'atteigne. Afin d'anticiper cette fois la prise d'un objet, on a décidé de faire atteindre la destination par le point2.

On utilise une méthode basé sur l'intersection de cercles (Voir dessin) :
Après avoir trouvé les deux points d'intersection, on calcule l'angle entre ces points et la destination par rapport à Point0 (point vers destination => Le signe de l'angle est très important).
Pour chacun des angles, on calcule la nouvelle position du point2 et on sélectionne l'intersection qui retourne le Point2 le plus proche de la destination.

On fait donc une rotation du bras1 suivant l'angle donné, la destination se retrouve ainsi sur le cercle de centre Point1 et de rayon la longueur du bras 2. Il nous reste plus qu'à calculer l'angle qui sépare le Point2 de la destination et à bouger le Bras 2 en conséquence. L'objectif est atteint.



4°) RECUPERATION D'UN OBJET

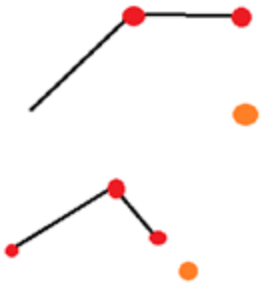
A présent, on est en mesure d'aller récupérer un objet.

Pour ce faire, on détermine un angle de prise de l'objet. Il y a donc deux endroits où le récupérer. On choisit celui qui est le plus proche du Point 2.

On détermine aussi un décalage pour faciliter la prise de l'objet par la pince. On a donc un nouveau point décalé qui conserve l'angle de prise. Ce point est la destination du Point2.

Une fois le robot déplacé, on effectue une rotation de l'angle du bras 3 pour que le point 3 soit aligné avec le point 2 et la destination

Le point2 avance ensuite vers l'objectif suivant la distance point3versObjectif.



C) CALCUL SUR LE PLAN xOz

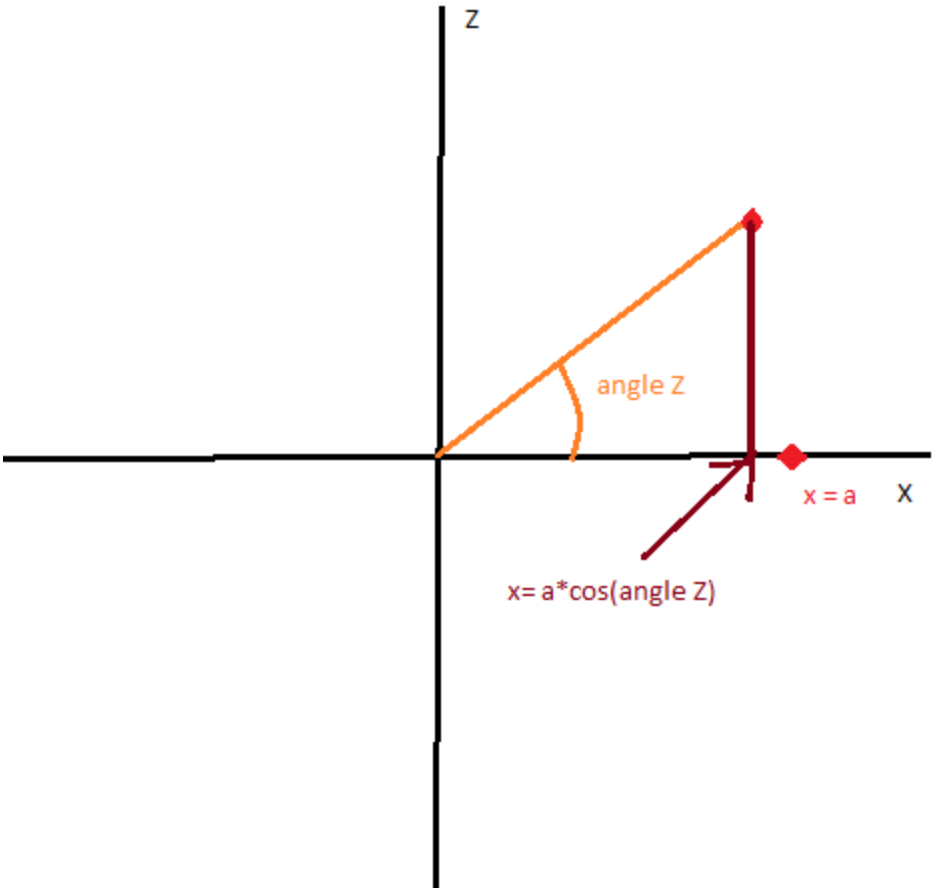
1°) ROTATION

Passons maintenant à la rotation sur le plan xOz. Elle s'effectue uniquement par rapport au point 0.
On commence par définir un angle de rotation sur xOz. Lorsque l'angle est à zéro, les points ont tous la même cote, comme définie plus haut.

On calcule les coordonnées des points comme suit :

XPoint1 = (xPoint1-xPoint0) * cos(angle) - (zPoint1-zPoint0) * sin(angle) +xPoint0
ZPoint1 = (xPoint1-xPoint0) * sin(angle) +(zPoint1-zPoint0) * sin(angle) +xPoint0

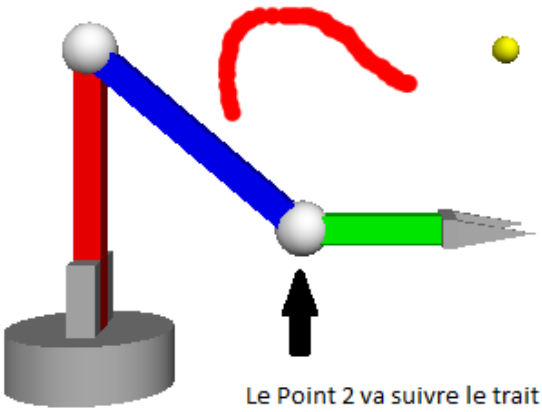
Cependant, lorsque l'angle de rotation n'est pas nul, il faut recalculer les abscisses que l'on avait définies dans le plan xOy.



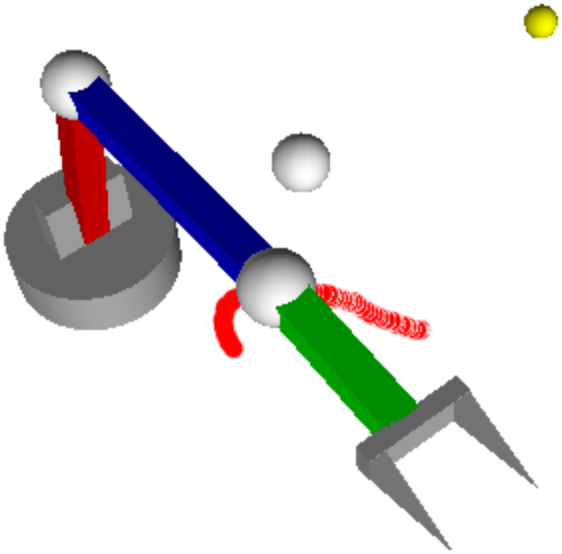
2°) DÉPLACEMENT DE LA PINCE VERS UNE DESTINATION

Pour que le point 2 puisse se déplacer vers une destination lorsque l'ange des Z n'est pas nulle, on a décidé de conserver les calculs de destination du plan xOy.
En gros, on cherche l'angle entre un point du robot lorsque l'angle Z est nul (sauf le point 0), et la destination A, par rapport au point 0.
On obtient ainsi en faisant une rotation inverse, une destination B de même z que les autres points du robot.
On calcule les angles de déplacement de la même manière que précédemment, puis on aligne le robot avec la destination A, sur le plan xOz.
Le fait d'appliquer les angles de déplacement, permet d'atteindre la destination A.

C'est pour cette raison que l'on a décidé de définir des coordonnées z identique pour chaque point du robot : cela permet d'effectuer les calculs d'angle sans prendre en compte les coordonnées z.



Le Point 2 va suivre le trait rouge



L'ensemble de points n'avaient pas la même z que les axes initiaux du robot.
Le robot a du s'aligner avec chaque point pour l'atteindre.
(Ne pas prendre en compte les deux sphères libre)

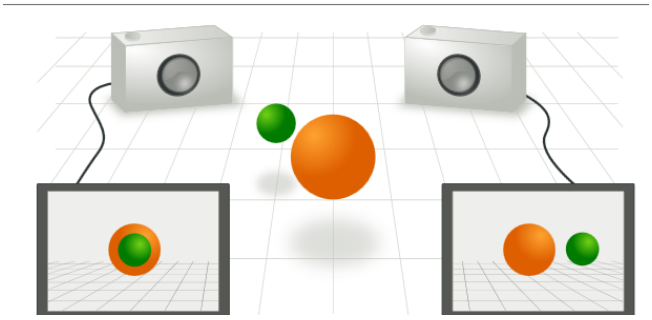
III- Ce qu'il reste à faire & Améliorations

- La première chose qui reste à faire est d'établir le lien entre le robot et le simulateur. Pour cela, il faut au préalable configurer des objets servo avec les attributs (Axe,ang mix max..) qui doivent correspondre aux différentes caractéristique des servomoteurs du robot.
  Ensuite, chaque angle de chaque axe est définit dans des variables dans le simulateur. Il suffit de les récupérer et de faire un commit (classe SSC-32) toute les x seconde. La variable x reste à définir, mais à notre avis elle se situerait entre 0.1 et 0.2 seconde. Ainsi à chaque mouvement du simulateur, le robot reçoit toutes les commandes nécessaires pour chaque servomoteur.

- On peut penser à créer une interface graphique complète prenant en compte toutes les caractéristiques du robot permettant de créer le simulateur.

- Une partie importante consiste à recevoir les ordres de mouvement pour centrer la caméra avec le marqueur. Il faut définir lequel des deux groupes travaillera dessus (Calculer les ordres). La connexion via client/serveur (Agents) entre nous est déjà établit.

- Une dernière amélioration possible est d'utiliser la géométrie épipolaire pour déterminer les coordonnées de l'objet. Cette technique consiste à utiliser deux caméras au lieu d'une pour pouvoir déterminer la profondeur de l'objet. (Même principe de fonctionnement des yeux ou bien des lunette 3D)



  Si le marqueur de l'objet a été vu par deux caméras différentes, on peut en fonction des positions des "trous" dans l'espace, et par rapport aux images, reconstituer les deux droites, et leur intersection, qui est le point objet, que l'on peut donc calculer ultérieurement. On a ainsi la reconstruction à trois dimensions d'un point objet dont on peut identifier les points en correspondance sur deux images

IV- Annexe

1. Les différentes classes utilisé pour l'interface de connexion.

Afin de mieux comprendre le fonctionnement de la communication avec la carte,
nous avons à notre disposition le manuel SSC-32 qui regroupe toutes les instructions nécessaires pour contrôler les différents servomoteurs, voici un rapide résumer des instructions disponibles :

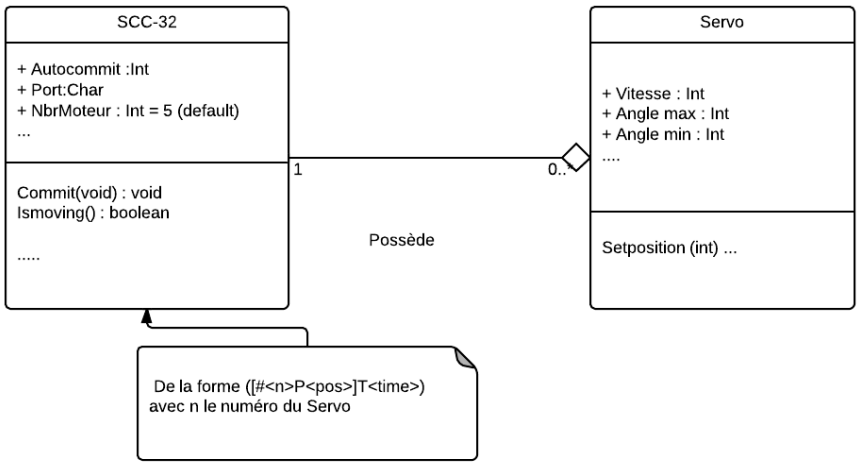
Command Types and Groups			
1	Servo Movement	7	Read Analog Inputs
2	Discrete Output	8	12 Servo Hexapod Gait Sequencer
3	Byte Output	9	Query Hex Sequencer
4	Query Movement Status	10	Get Version
5	Query Pulse Width	11	Go to Boot
6	Read Digital Inputs	12	MiniSSC-II Compatibility

Servo Move or Group Move.

# <ch> P <pw> S <spd> ... # <ch> P <pw> S <spd> T <time> <cr>	
<ch>	Channel number in decimal, 0-31
<pw>	Pulse width in microseconds, 500-2500
<spd>	Movement speed in uS per second for one channel (Optional)
<time>	Time in mS for the entire move, affects all channels, 65535 max (Optional)
<cr>	Carriage return character, ASCII 13 (Required to initiate action)
<esc>	Cancel the current action, ASCII 27

Par exemple : Si on envoie en ASCII la commande suivante: "#5 P1600 S100 <cr>"
Cette commande va faire bouger le servomoteur connecté au Chanel n°5 à la position « 1600 ». La position est définit comme la durée des impulsions qui détermine l'angle absolu de l'axe de sortie et donc la position du bras de commande du servomoteur. Le tout exécuté à une vitesse définit à 752 us.
Pour mieux comprendre comment la vitesse est définie, on considère qu'a 1000uS il est effectué en 1 seconde une rotation de 90°. Donc pour une valeur de la vitesse à 100 uS signifie que le servomoteur va prendre 10 seconde pour faire une rotation à 90°.
De plus, il existe des commandes qui permettent de recevoir des informations provenant de la carte. Par exemple si on l'envoie la commande : "Q <cr>" renvoie "+" si le robot est en mouvement.

Donc à partir des ces informations et des poste dans le forum du créateur de la carte, nous avons en python deux classes :



La première classe SSC-32 regroupe des méthode permettant d'établir la connexion et d'envoyer les différentes commandes à la carte. (Socket, port,).
Celle-ci possède plusieurs objet Servo. Un servomoteur est définie par plusieurs attribut (Position ,vitesse,angle min, angle max...).

Pour bien comprendre, nous allons prendre un exemple :

Tout d'abord on initialise la connexion USB, généralement le port est « /dev/ttyUSB1 ». Le 115200 représente le braudbit(la vitesse de connexion).
ssc = ssc32.SSC32('/dev/ttyUSB1', 115200 ,vitesse=1000)

On initialise nos objets servomoteurs grâce à la fonction __getitem__ :
Servo1 = ssc['Servo1']
Servo2 = ssc['Servo2']
…

Pour cette exemple, les valeurs d'angle min,max... sont par défaut celle écrit dans le code.
On définit une liste de chaque servomoteur avec la position en degrés voulust o
position = [(Servo1, 50),
(Servo2, -130),
…]

def ChargerPosition(listePosition):

for move in listePosition:

move[0].degree(move[1]) . # degree
ssc.commit(600) # 600 est le temps voulus.
While (ssc.IsMoving){ # tant que le robot est en mouvement
time.sleep(0.3) }

#On temporise pour ne pas exécuter toutes les commandes en même temps.

Une fois les valeurs chargées, il suffit d'utiliser la fonction commit() qui va envoyé pour chaque servo la commande suivante :*[#<n>P<pos>]T<time>.*